DSM Working Group: Composition and Integration of DSMs

Contributions From

Tony Clark
Jeff Gray
Danny Groenewegen
David Heise
Heiko Kattenstoth
Zekai Deminezen
Arturo Sanchez
Dirk Reiss
Jendrik Johannes
Thomas Kuhn
Arnon Sturm
Martin Schindler

Issues

- No explicit definition of interfaces. Don’t know which elements to extend, the semantics might break.
- Stay within the language, migrate to the new model is a problem. Possible solution: patching problems, new version contains patching modules, contains the deltas.
- Patches might be difficult due to links to the meta-levels which change. Can ship transformations with the changes. Difficult to do.

Problems

- When a DSL is designed, don't have extensibility in mind. One shot solution. A Solution might be to identify the variation points at the beginning. Problem: how to build extensibility into a language.
- Extensibility via interfaces – how to do that in language design.
- Example from WebDSL: attaching behaviour to text, was not envisaged originally need to go into the code and add this.

Language Extension Strategies

- Principles for designing-in extensibility.
- Monticore holes in the grammar – extension points for syntax. Editors are defined separately then combined at run-time in to the tool for the language. Also for code generation.
- UML left open the action language – OMG left a placeholder for action languages. To do this need a component and interface model for combining language elements.

Language Decomposition and Extensions

- Is it possible to have a standard library for example an expression language that can be reused in new languages. For example, OCL has been decomposed like this.
- Extension Strategies:
  - Use 1 language and then adapt it to produce a new language. For example UML profiles.
  - Use several languages (could be independent) and then merge.
  - A (possibly incomplete) language with extension points.
- Should a language know that it is being extended?

Semantics

- When integrating multiple independent languages there are problems due to semantic integration. Often the semantics is informal and cannot easily be combined.
- Need a way to standardize on the language semantics to facilitate language combination.
- Approaches:
  - Common language
  - Pair to pair transformations
  - Proxies or middleware
- An approach: use proxies over independent language definitions. E.g. C++ with templates and Java.
- Semantics is key to language integration.
- Possibly project the semantics of languages into a common representation to perform combination.
- Common meta-meta-models have been tried before. Perhaps agreeing on such a thing would be a solution (the MOF++ approach).

Integration

- Integration approaches:
  - One language references another
  - Java syntax embedded in the new language
  - Common Semantics.
    - Code generator
    - Interpreter.
- Take 2 independent languages and construct a meta-model that covers both for integration.
- Adding a new language must involve the constraints and requirements that must be met by composition.
- Be clear about the requirements for combination: do you want a single tool for the merged language or continue with the original tool-set.
- Are there patterns for language integration that can be identified and documented?
- Currently integration is performed manually. Can language components be integrated automatically? What does this requirement mean for the models used to express language interfaces etc.
- Problem: integrating many special purpose languages can end up with a single GPL.

Language Components and Interfaces

- Problem: what does an interface for a language look like?
- Problem: There is no component model for languages.
- Start with a base module and define the interfaces for future extensions.
- Languages are defined as a core and extension modules.
- Interfaces should include requirements and guarantees for use and combination of the language component.
- Example: build an expression language, plug-in the expression language to several new languages. Question: what interface should the expression language have to support insertion and combination.
- Aim: How to achieve a product-line approach to language definition.
- Type systems are going to be part of the interface of a language component. Type systems need to be integrated (or mapped) when language components are integrated.
- Problem: When defining interfaces how to anticipate all possible future extensions and combinations of the language.

Language Tools

- Problem: how to combine existing tools (compilers, editors etc) when combining existing languages into a new DSL.
- How to reuse tools when defining a modification of an existing language.

Guidelines

- DSL designers need to be aware of where the extension points are.
- Possible to leave plug-points in a language.
- Look to produce and use a library of reusable components.
- Standardize and define the semantics for language combination.
- Common Meta-models makes life easier.
- Designing language components for reuse is easier than reusing an existing language.
- The abstract syntax and the semantics seems core the notion of languages and integration.